QingKeV4 Microprocessor Manual

V1.5

Overview

QingKe V4 series microprocessors are self-developed 32-bit general-purpose MCU microprocessors based on standard RISC-V instruction set architecture. According to different application scenarios and instruction set combinations, this series includes V4A, V4B, V4C, V4J, V4F. The V4 series all support RV32IMAC instruction set extensions, of which V4F supports single-precision hardware floating-point, i.e., it supports RV32IMACF extensions. V4B, V4C, V4J, V4F also support custom extensions XW, in addition to Hardware Prologue/Epilogue (HPE), Vector Table Free (VTF), streamlined 1/2-wire debugging interface, support for the "WFE" instruction, physical memory protection (PMP), and other features, the V4J also supports instruction caching. V4J also supports instruction cache.

Features

Features	Description
ISA	RV32IMAC[F]
Pipeline	Level 3
FPU	Supports single-precision floating-point
Branch prediction	BHT/BTB/RAS
Instruction cache	Support up to 64KB
Interrupt	Supports a total of 256 interrupts including exceptions, and supports VTF
HPE	Supports up to 3 levels of HPE
PMP	Supports 4 memory protection zones
Low-power consumption mode	Supports Sleep and Deep sleep modes, and support WFI and WFE sleep methods
Extended instruction set	Supports half-word and byte operation compression instructions
Debug	1/2-wire SDI, standard RISC-V debug

Chapter 1 Overview

QingKe V4 series microprocessors include V4A, V4B, V4C and V4F, and there are certain differences between each series according to the applications, and the specific differences are detailed in Table 1-1.

Table 1-1 Microprocessor comparison overview

Feature / Model	ISA	HPE number of levels	Interruptions nesting number of levels	VTF number of channels	Pipeline	Vector table mode	Cache	Extended Instruction (XW)	Number of memory protection areas
V4A	RV32IMAC	2	2	4	3	Address/ Instruction	×	×	4
V4B	RV32IMAC	2	2	4	3	Address/ Instruction	×	√	0
V4C	RV32IMAC	2	2	4	3	Address/ Instruction	×	√	4
V4F	RV32IMACF	3	8	4	3	Address/ Instruction	×	√	4
V4J	RV32IMAC	2	2	4	3	Address/ Instruction	I-Cache	√	4

Note: OS task switching generally uses stack push, which are not limited in number of levels.

1.1 Instruction Set

QingKe V4 series microprocessors follow the standard RISC-V Instruction Set Architecture (ISA). Detailed documentation of the standard can be found in "The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Document Version 2.2" on the RISC-V International website. The RISC-V instruction set has a simple architecture and supports a modular design, allowing for flexible combinations based on different needs, and the V4 series supports the following instruction set extensions.

- RV32: 32-bit architecture, general-purpose register bit width of 32 bits
- I: Supports shaping operation, with 32 shaping registers
- M: Supports shaping multiplication and division instructions
- A: Support atomic instructions, simplify lr and sc instructions, and only execute them as LW and SW, and the return of sc results is always successful.
- C: Supports 16-bit compression instruction
- F: Support single-precision floating-point operation, with 32 floating-point registers.
- XW: 16-bit compression instruction for self-expanding byte and halfword operations.

Note: 1: The sub-instruction sets supported by different models may be different, for details, please refer to Table 1-1.

2: To further improve code density, extend the XW subset by adding the following compression directives c.lbu/c.lhu/c.sb/c.sh/c.lbusp/c.lhusp/c.sbsp/c.shsp, use based on the MRS compiler or the toolchain it provides.

1.2 Register Set

The RV32I has 32 register sets from x0-x31. The V3 series does not support the "F" extension, i.e., there is no floating-point register set. In the RV32, each register is 32 bits. Table 1-2 below lists the registers of RV32I and their descriptions.

Table 1-2 RISC-V registers

Register	ABI Name	Description	Storer
x0	zero	Hardcoded 0	-
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	-
x4	tp	Thread pointer	-
x5-7	t0-2	Temporary register	Caller
x8	s0/fp	Save register/frame pointer	Callee
x9	s1	Save register	Callee
x10-11	a0-1	Function parameters/return values	Caller
x12-17	a2-7	Function parameters	Caller
x18-27	a2-11	Save register	Callee
X28-31	t3-6	Temporary register	Caller
f0-7	ft0-7	Floating-point temporary register	Caller
f8-9	fs0-1	Floating-point save register	Callee
f10-11	fa0-1	Floating-point function parameters/return values	Caller
f12-17	fa2-7	Floating-point function parameters	Caller
f18-27	fs2-11	Floating-point save register	Callee
f28-31	ft8-11	Floating-point temporary register	Caller

The Caller attribute in the above table means that the called procedure does not save the register value, and the Callee attribute means that the called procedure saves the register.

1.3 Privilege Mode

The standard RISC-V architecture includes three privileged modes: Machine mode, Supervisor mode, and User mode, as shown in Table 1-3 below. The machine mode is a mandatory mode, and the other modes are optional modes. For details, you can refer to "The RISC-V Instruction Set Manual Volume II: Privileged Architecture", which can be downloaded for free from the RISC-V International website.

Table 1-3 RISC-V architecture privilege mode

Code	Name	Abbreviations
0b00	User Mode	U
0b01	Supervisor Model	S
0b10	Reserved	Reserved
0b11	Machine mode	M

QingKe V4 series microprocessors support two of these privileged modes.

- Machine mode

Machine mode has the highest privileges, and this mode allows the program to access all Control and Status Registers (CSRs), as well as all physical memory protection units except for the Physical Memory Protection (PMP) lock. The power-up default is in Machine mode. When the execution of mret (machine mode return instruction) returns, according to the MPP bit in the CSR register mstatus (machine mode status register), if MPP=0b00, it will exit Machine mode and enter User mode, and if MPP=0b11, it will remain in Machine mode.

- User mode

User mode has the lowest privilege, and only limited CSR registers and physical address areas allowed by

PMP privilege. When an exception or interrupt occurs, the microprocessor goes from User mode to Machine mode to handle exceptions and interrupts.

1.4 CSR Register

A series of CSR registers are defined in the RISC-V architecture to control and record the operating state of the microprocessor. These CSRs can be extended by 4096 registers using an internal dedicated 12-bit address coding space. And use the high two CSR[11:10] to define the read/write permission of this register, 0b00, 0b01, 0b10 for read/write allowed and 0b11 for read only. Use the two bits CSR[9:8] to define the lowest privilege level that can access this register, and the value corresponds to the privilege mode defined in Table 1-3. The QingKe V4 series microprocessors include the standard definition of relevant CSR registers in addition to some custom CSR registers extended for control and status logging of enhanced functions. The CSR registers implemented by the microprocessor are detailed in Chapter 8.

Chapter 2 Exception

Exception mechanism, which is a mechanism to intercept and handle "unusual operation events". QingKe V4 series microprocessors are equipped with an exception response system that can handle up to 256 exceptions, including interrupts. When an exception or interruption occurs, the microprocessor can quickly respond and handle the exception and interruption events.

2.1 Exception Types

The hardware behavior of the microprocessor is the same whether an exception or an interrupt occurs. The microprocessor suspends the current program, moves to the exception or interrupt handler, and returns to the previously suspended program when processing is complete. Broadly speaking, interrupts are also part of exceptions. Whether exactly the current occurrence is an interrupt or an exception can be viewed through the Machine mode exception cause register mcause. The mcause[31] is the interrupt field, which is used to indicate whether the cause of the exception is an interrupt or an exception. mcause[31]=1 means interrupt, mcause[31]=0 means exception. mcause[30:0] is the exception code, which is used to indicate the specific cause of the exception or the interrupt number, as shown in the following table.

Table 2-1 V4 microprocessor exception codes

Interrupt	Exception codes	Synchronous / Asynchronous	Reason for exception
1	0-1	-	Reserved
1	2	Precise asynchronous	NMI interrupts
1	3-11	-	Reserved
1	12	Precise asynchronous	SysTick interrupts
1	13	-	Reserved
1	14	Synchronous	Software interrupts
1	15	-	Reserved
1	16-255	Precise asynchronous	External interrupt 16-255
0	0	Synchronous	Instruction address misalignment
0	1	Synchronous	Fetch command access error
0	2	Synchronous	Illegal instructions
0	3	Synchronous	Breakpoints
0	4	Synchronous	Load instruction access address misalignment
0	5	Non- precision asynchronous	Load command access error
0	6	Synchronous	Store/AMO instruction access address misalignment
0	7	Non-precision asynchronous	Store/AMO command access error
0	8	Synchronous	Environment call in User mode
0	11	Synchronous	Environment call in Machine mode

'Synchronous' in the table means that an instruction can be located exactly where it is executed, such as an ebreak or ecall instruction, and each execution of that instruction will trigger an exception. 'Asynchronous' means that it is not possible to pinpoint an instruction, and the instruction PC value may be different each time an exception occurs. 'Precise asynchronous' means that an exception can be located exactly at the boundary of an instruction, i.e., the state after the execution of an instruction, such as an external interrupt. 'Non- precision asynchronous' means that the boundary of an instruction cannot be precisely located, and

may be the state after an instruction has been interrupted halfway through execution, such as a memory access error. Access to memory takes time, and the microprocessor usually does not wait for the end of the access when accessing memory, but continues to execute the instruction, when the access error exception occurs again, the microprocessor has already executed the subsequent instructions, and cannot be precisely located.

2.2 Entering Exception

When the program is in the process of normal operation, if for some reason, triggered into an exception or interrupt. The hardware behavior of the microprocessor at this point can be summarized as follows.

(1) Suspend the current program flow and move to the execution of exception or interrupt handling functions.

The entry base address and addressing mode of the exception or interrupt function are defined by the exception entry base address register mtvec. mtvec[31:2] defines the base address of the exception or interrupt function. mtvec[1:0] defines the addressing mode of the handler function, where mtvec[0] defines the entry mode of the exception and interrupt. when mtvec[0]=0, all exceptions and interrupts use When mtvec[0]=0, all exceptions and interrupts use a unified entry, i.e., when an exception or interrupt occurs, it turns to the base address defined by mtvec[31:2] for execution. When mtvec[0]=1, exceptions and interrupts use vector table mode, i.e., each exception and interrupt is numbered, and the address is shifted according to interrupt number*4, and when an exception or interrupt occurs, it is shifted to the base address defined by mtvec[31:2] + interrupt number*4 for execution. The vector mode mtvec[1] defines the identification mode of the vector table. When mtvec[1]=0, the instruction stored at the vector table is an instruction to jump to the exception or interrupt handling function, or it can be another instruction; when mtvec[1]=1, the absolute address of the exception handling function is stored at the vector table.

(2) Update CSR register

When an exception or interrupt is entered, the microprocessor automatically updates the relevant CSR registers, including the machine mode exception cause register mcause, the machine mode exception pointer register mepc, the Machine mode exception value register mtval, and the Machine mode status register mstatus.

- Update mcause

As mentioned before, after entering an exception or interrupt, its value reflects the current exception type or interrupt number, and the software can read this register value to check the cause of the exception or determine the source of the interrupt, as detailed in Table 2-1.

- Update mepc

The standard definition of the return address of the microprocessor after exiting an exception or interrupt is stored in mepc. So when an exception or interrupt occurs, the hardware automatically updates the mepc value to the current instruction PC value when the exception is encountered, or the next pre-executed instruction PC value before the interrupt. After the exception or interrupt is processed, the microprocessor uses its saved value as the return address to return to the location of the interrupt to continue execution.

However, it is worth noting that.

1. mepc is a readable and writable register, and the software can also modify the value for the purpose of modifying the location of the PC pointer running after the return.

2. When an interrupt occurs, i.e., when the exception cause register mcause[31]=1, the value of mepc is updated to the PC value of the next unexecuted instruction at the time of the interrupt.
3. And when an exception occurs, the value of mepc is updated to the instruction PC value of the current exception when the exception cause register mcause[31]=0. So at this time when the exception returns, if we return directly using the value of mepc, we still continue to execute the instruction that generated the exception before, and at this time, we will continue to enter the exception. Usually, after we handle the exception, we can modify the value of mepc to the value of the next unexecuted instruction and then return. For example, if we cause an exception due to ecall/ebreak, after handling the exception, since ecall/ebreak (c.ebreak is 2 bytes) is a 4-byte instruction, we only need the software to modify the value of mepc to mepc+4 (c.ebreak is mepc+2) and then return.

- Update mtval

When exceptions and interrupts are entered, the hardware will automatically update the value of mtval, which is the value that caused the exception. The value is typically.

1. If an exception is caused by a memory access, the hardware will store the address of the memory access at the time of the exception into mtval.
2. If the exception is caused by an illegal instruction, the hardware will store the instruction code of the instruction into mtval.
3. If the exception is caused by a hardware breakpoint, the hardware will store the PC value at the breakpoint into mtval.
4. For other exceptions, the hardware sets the value of mtval to 0, such as ebreak, the exception caused by ecall instruction.
5. When entering the interrupt, the hardware sets the value of mtval to 0.

- Update mstatus

Upon entering exceptions and interrupts, the hardware updates certain bits in mstatus.

1. MPIE is updated to the MIE value before entering the exception or interrupt, and MPIE is used to restore the MIE after the exception and interrupt are over.
2. MPP is updated to the privileged mode before entering exceptions and interrupts, and after the exceptions and interrupts are over, MPP is used to restore the previous privileged mode.
3. QingKe V4 microprocessor supports interrupt nesting in Machine mode, and MIE will not be cleared after entering exceptions and interrupts.

(3) Update microprocessor privilege mode

When exceptions and interrupts occur, the privileged mode of the microprocessor is updated to Machine mode.

2.3 Exception Handling Functions

Upon entering an exception or interrupt, the microprocessor executes the program from the address and mode defined by the mtvec register. When using the unified entry, the microprocessor takes a jump instruction from the base address defined by mtvec[31:2] based on the value of mtvec[1], or gets the exception and interrupt handling function entry address and goes to execute it instead. At this time, the exception and interrupt handling function can determine whether the cause is an exception or interrupt based on the value of mcause[31], and the type and cause of the exception or the corresponding interrupt can be judged by the exception code and handled accordingly.

When using the base address + interrupt number *4 for offset, the hardware automatically jumps to the vector table to get the entry address of the exception or interrupt function based on the interrupt number and

jumps to execute it.

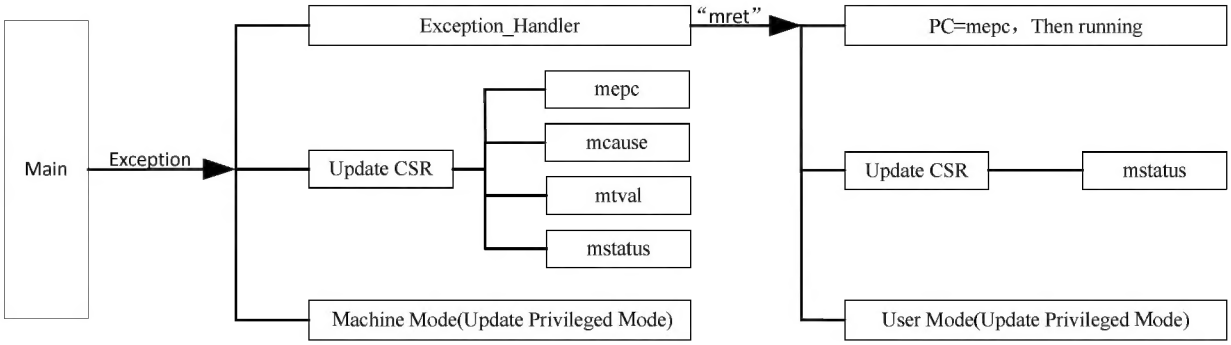
2.4 Exception Exit

After the exception or interrupt handler is completed, it is necessary to exit from the service program. After entering exceptions and interrupts, the microprocessor enters Machine mode from User mode, and the processing of exceptions and interrupts is also completed in Machine mode. When it is necessary to exit exceptions and interrupts, it is necessary to use the mret instruction to return. At this time, the microprocessor hardware will automatically perform the following operations.

- The PC pointer is restored to the value of CSR register mepc, i.e., execution starts at the instruction address saved by mepc. It is necessary to pay attention to the offset operation of mepc after the exception handling is completed.
- Update CSR register mstatus, MIE is restored to MPIE, and MPP is used to restore the privileged mode of the previous microprocessor.

The entire exception response process can be described by the following Figure 2-1.

Figure 2-1 Exception response process diagram



Chapter 3 PFIC and Interrupt Control

QingKe V4 microprocessor is designed with a Programmable Fast Interrupt Controller (PFIC) that can manage up to 256 interrupts including exceptions. The first 16 of them are fixed as internal interrupts of the microprocessor, and the rest are external interrupts, i.e. the maximum number of external interrupts can be extended to 240. Its main features are as follows.

- 240 external interrupts, each interrupt request has independent trigger and mask control bits, with dedicated status bits
- Programmable multi-level interrupt nesting, maximum nesting depth 8 levels
- Special fast interrupt in/out mechanism, hardware automatic stacking and recovery, maximum hardware stacking depth of 3 levels, no instruction overhead
- Vector Table Free (VTF) interrupt response mechanism, 4-channel programmable direct access to interrupt vector addresses

Note: The maximum nesting depth and HPE depth supported by interrupt controllers vary for different microprocessor models, which can be found in Table 1-1.

The vector table of interrupts and exceptions is shown in Table 3-1 below.

Table 3-1 Exception and interrupt vector table

Number	Priority	Type	Name	Description
0	-	-	-	-
1	-	-	-	-
2	-5	Fixed	NMI	Non-maskable interrupt
3	-4	Fixed	EXC	Exception interrupt
4	-	-	-	-
5	-3	Fixed	ECALL-M	Machine mode callback interrupt
6-7	-	-	-	-
8	-2	Fixed	ECALL-U	User mode callback interrupt
9	-1	Fixed	BREAKPOINT	Breakpoint callback interrupt
10-11	-	-	-	-
12	0	Programmable	SysTick	System timer interrupt
13	-	-	-	-
14	1	Programmable	SWI	Software interrupt
15	-	-	-	-
16-255	2-241	Programmable	External interrupt	External interrupt 16-255

3.1 PFIC Register Set

Table 3-2 PFIC Registers

Name	Access address	Access	Description	Reset value
PFIC_ISRx	0xE000E000 -0xE000E01C	RO	Interrupt enable status register x	0x00000000
PFIC_IPRx	0xE000E020 -0xE000E03C	RO	Interrupt pending status register x	0x00000000
PFIC_ITHRESDR	0xE000E040	RW	Interrupt priority threshold	0x00000000

			configuration register	
PFIC_CFGR	0xE000E048	RW	Interrupt configuration register	0x00000000
PFIC_GISR	0xE000E04C	RO	Interrupt global status register	0x00000000
PFIC_VTFADDRRx	0xE000E060 -0xE000E06C	RW	VTF x offset address register	0x00000000
PFIC_IENRx	0xE000E100 -0xE000E11C	WO	Interrupt enable setting register x	0x00000000
PFIC_IRERx	0xE000E180 -0xE000E19C	WO	Interrupt enable clear register x	0x00000000
PFIC_IPSRx	0xE000E200 -0xE000E21C	WO	Interrupt pending setting register x	0x00000000
PFIC_IPRRx	0xE000E280 -0xE000E29C	WO	Interrupt pending clear register x	0x00000000
PFIC_IACTRx	0xE000E300 -0xE000E31C	RO	Interrupt activation status register x	0x00000000
PFIC_IPRIORx	0xE000E400 -0xE000E43C	RW	Interrupt priority configuration register	0x00000000
PFIC_SCTLR	0xE000ED10	RW	System control register	0x00000000

Note: 1. The default value of PFIC_ISR0 register is 0xC, which means that NMI and exception are always enabled by default.

2.ECALL-M, ECALL-U, BREAKPOINT are all a case of EXC, the status is indicated by the status bit 3 of EXC.

3. NMI and EXC support interrupt pending clear and setup operation, but not interrupt enable clear and setup operation.

4.ECALL-M, ECALL-U, BREAKPOINT do not support interrupt pending clear and set, interrupt enable clear and set operation.

Each register is described as follows.

Interrupt Enable Status and Interrupt Pending Status Registers (PFIC_ISR<0-7>/PFIC_IPR<0-7>)

Name	Access address	Access	Description	Reset value
PFIC_ISR0	0xE000E000	RO	Interrupt 0-31 enable status register, a total of 32 status bits [n], indicating #n interrupt enable status *Note: NMI and EXC are enabled by default*	0x0000000C
PFIC_ISR1	0xE000E004	RO	Interrupt 32-63 enable status register, total 32 status bits	0x00000000
…	…	…	…	…
PFIC_ISR7	0xE000E01C	RO	Interrupt 224-255 enable status register, total 32 status bits	0x00000000
PFIC_IPR0	0xE000E020	RO	Interrupt 0-31 pending status register, a total of 32 status bits [n], indicating the pending status	0x00000000

			of interrupt #n	
PFIC_IPR1	0xE000E024	RO	Interrupt 32-63 pending status registers, 32 status bits in total	0x00000000
…	…	…	…	…
PFIC_IPR7	0xE000E03C	RO	Interrupt 244-255 pending status register, 32 status bits in total	0x00000000

2 sets of registers are used to enable and de-enable the corresponding interrupts.

Interrupt Enable Setting and Clear Registers (PFIC_IENR<0-7>/PFIC_IRER<0-7>)

Name	Access address	Access	Description	Reset value
PFIC_IENR0	0xE000E100	WO	Interrupt 0-31 enable setting register, a total of 32 setting bits [n], for interrupt #n enable setting *Note: NMI and EXC are enabled by default*	0x00000000
PFIC_IENR1	0xE000E104	WO	Interrupt 32-63 enable setting register, total 32 setting bits	0x00000000
…	…	…	…	…
PFIC_IENR7	0xE000E11C	WO	Interrupt 224-255 enable setting register, total 32 setting bits	0x00000000
-	-	-	-	-
PFIC_IRER0	0xE000E180	WO	Interrupt 0-31 enable clear register, a total of 32 clear bits [n], for interrupt #n enable clear *Note: NMI and EXC cannot be operated*	0x00000000
PFIC_IRER1	0xE000E184	WO	Interrupt 32-63 enable clear register, total 32 clear bits	0x00000000
…	…	…	…	…
PFIC_IRER7	0xE000E19C	WO	Interrupt 244-255 enable clear register, total 32 clear bits	0x00000000

2 sets of registers are used to enable and de-enable the corresponding interrupts.

Note: When using registers to mask any interrupt or using CSR registers to mask global interrupts, add a 'fence.i' instruction to synchronize between core control state and interrupt enable state.

Interrupt Pending Setting and Clear Registers (PFIC_IPSR<0-7>/PFIC_IPRR<0-7>)

Name	Access address	Access	Description	Reset value
PFIC_IPSR0	0xE000E200	WO	Interrupt 0-31 pending setting register, 32 setting bits [n], for interrupt #n pending setting *Note: ECALL-M, ECALL-U, BREAKPOINT do not support this operation.*	0x00000000

PFIC_IPSR1	0xE000E204	WO	Interrupt 32-63 pending setup register, total 32 setup bits	0x00000000
…	…	…	…	…
PFIC_IPSR7	0xE000E21C	WO	Interrupt 224-255 pending setting register, 32 setting bits in total	0x00000000
-	-	-	-	-
PFIC_IPRR0	0xE000E280	WO	Interrupt 0-31 pending clear register, a total of 32 clear bits [n], for interrupt #n pending clear *Note: ECALL-M, ECALL-U, BREAKPOINT do not support this operation.*	0x00000000
PFIC_IPRR1	0xE000E284	WO	Interrupt 32-63 pending clear register, total 32 clear bits	0x00000000
…	…	…	…	…
PFIC_IPRR7	0xE000E29C	WO	Interrupt 244-255 pending clear register, total 32 clear bits	0x00000000

When the microprocessor enables an interrupt, it can be set directly through the interrupt pending register to trigger into the interrupt. Use the interrupt pending clear register to clear the pending trigger.

Interrupt Activation Status Register (PFIC_IACTR<0-7>)

Name	Access address	Access	Description	Reset value
PFIC_IACTR0	0xE000E300	RO	Interrupt 0-31 activates the status register with 32 status bits [n], indicating that interrupt #n is being executed	0x00000000
PFIC_IACTR1	0xE000E304	RO	Interrupt 32-63 activation status registers, 32 status bits in total	0x00000000
…	…	…	…	…
PFIC_IACTR7	0xE000E31C	RO	Interrupt 224-255 activation status register, total 32 status bits	0x00000000

Each interrupt has an active status bit that is set up when the interrupt is entered and cleared by hardware when mret returns.

Interrupt Priority and Priority Threshold Registers (PFIC_IPRIOR<0-7>/PFIC_ITHRESDR)

Name	Access address	Access	Description	Reset value
PFIC_IPRIOR0	0xE000E400	RW	Interrupt 0 priority configuration. For V4A: [7:4]: Priority control bits If the configuration is not nested, no preemption bit If configured with 2 levels of nesting, bit7 is the preempted bit.	0x00

			[3:0]: Reserved, fixed to 0 For V4B/C: [7:5]: Priority control bits If the configuration is not nested, no preemption bit If configured with 2 levels of nesting, bit7 is the preempted bit. [4:0]: Reserved, fixed to 0 For V4F: [7:5]: Priority control bits If the configuration is not nested, no preemption bit If configured with 2 levels of nesting, bit7 is the preempted bit. If configured with 4 levels of nesting, bit7-bit6 is the preempted bit. If configured with 8 levels of nesting, bit7-bit5 is the preempted bit. [4:0]: Reserved, fixed to 0 *Note: The smaller the priority value, the higher the priority. If the same preemption priority interrupt hangs at the same time, the interrupt with the higher priority will be executed first.*	
PFIC_IPRIOR1	0xE000E401	RW	Interrupt 1 priority setting, same function as PFIC_IPRIOR0	0x00
PFIC_IPRIOR2	0xE000E402	RW	Interrupt 2 priority setting, same function as PFIC_IPRIOR0	
…	…	…	…	…
PFIC_IPRIOR254	0xE000E4FE	RW	Interrupt 254 priority setting, same function as PFIC_IPRIOR0	0x00
PFIC_IPRIOR255	0xE000E4FF	RW	Interrupt 255 priority setting, same function as PFIC_IPRIOR0	0x00
-	-	-	-	-
PFIC_ITHRESDR	0xE000E040	RW	Interrupt priority threshold setting For V4A: [31:8]: Reserved, fixed to 0 [7:4]: Priority threshold [3:0]: Reserved, fixed to 0	0x00

			For V4B/C/F: [31:8]: Reserved, fixed to 0 [7:5]: Priority threshold [4:0]: Reserved, fixed to 0 *Note: For interrupts with priority value ≥ threshold, the interrupt service function is not executed when a hang occurs, and when this register is 0, it means the threshold register is invalid.*	

Interrupt Configuration Register (PFIC_CFGR)

Name	Access address	Access	Description	Reset value
PFIC_CFGR	0xE000E048	RW	Interrupt configuration register	0x00000000

Its folks are defined as:

Bit	Name	Access	Description	Reset value
[31:16]	KEYCODE	WO	Corresponding to different target control bits, the corresponding security access identification data needs to be written simultaneously in order to be modified, and the readout data is fixed to 0. KEY1 = 0xFA05; KEY2 = 0xBCAF; KEY3 = 0xBEEF.	0
[15:8]	Reserved	RO	Reserved	0
7	SYSRESET	WO	System reset (simultaneous writing to KEY3). Auto clear 0. Writing 1 is valid, writing 0 is invalid. *Note: Same function as the PFIC_SCTLR register SYSRESET bit.*	0
[6:0]	Reserved	RW	Reserved	0

V4 series microprocessors This register is mainly used for compatible.

Interrupt Global Status Register (PFIC_GISR)

Name	Access address	Access	Description	Reset value
PFIC_CFGR	0xE000E04C	RO	Interrupt global status register	0x00000000

Its folks are defined as:

Bit	Name	Access	Description	Reset value
[31:10]	Reserved	RO	Reserved	0
9	GPENDSTA	RO	Whether an interrupt is currently pending. 1: Yes; 0: No.	0
8	GACTSTA	RO	Whether an interrupt is currently being executed. 1: Yes; 0: No.	0

[7:0]	NESTSTA	RO	Current interrupt nesting status, currently supports a maximum of 8 levels of nesting, the maximum hardware stack depth is 3. If the nesting depth is set greater than 3, the lower three levels of interrupts should be configured for hardware stacking, and the remaining high priority levels use stack push. 0xFF: in level 8 interrupt. 0x7F: in level 7 interrupt. 0x3F: in level 6 interrupt. 0x1F: in level 5 interrupts. 0x0F: in level 4 interrupt. 0x07: in level 3 interrupt. 0x03: in level 2 interrupt. 0x01: in level 1 interrupt. 0x00: no interrupts occur. Other: Impossible situation. *Note: Cases greater than level 2 are only valid for V4F.*	0

VTF ID and Address Registers (PFIC_VTFIDR/PFIC_VTFADDRR<0-1>)

Name	Access address	Access	Description	Reset value
PFIC_VTFIDR	0xE000E050	RW	[31:24]: number of VTF 3 [23:16]: number of VTF 2 [15:8]: number of VTF 1 [7:0]: number of VTF 0	0x00000000
-	-	-	-	-
PFIC_VTFADDRR0	0xE000E060	RW	[31:1]: VTF 0 address, two-byte alignment [0]: 1: Enable VTF 0 channel 0: Close	0x00000000
PFIC_VTFADDRR1	0xE000E064	RW	[31:1]: VTF 1 address, two-byte alignment [0]: 1: Enable VTF 1 channel 0: Close	0x00000000

System Control Register (PFIC_SCTLR)

Name	Access address	Access	Description	Reset value
PFIC_SCTLR	0xE000ED10	RW	System control register	0x00000000

Each of them is defined as follows.

Bit	Name	Access	Description	Reset value

31	SYSRESET	WO	System reset, clear 0 automatically. write 1 valid, write 0 invalid, same effect as PFIC_CFGR register	0
[30:6]	Reserved	RO	Reserved	0
5	SETEVENT	WO	Set the event to wake up the WFE case.	0
4	SEVONPEND	RW	When an event occurs or interrupts a pending state, the system can be woken up from after the WFE instruction, or if the WFE instruction is not executed, the system will be woken up immediately after the next execution of the instruction. 1: Enabled events and all interrupts (including unenabled interrupts) can wake up the system. 0: Only enabled events and enabled interrupts can wake up the system.	0
3	WFITOWFE	RW	Execute the WFI command as if it were a WFE. 1: treat the subsequent WFI instruction as a WFE instruction. 0: No effect.	0
2	SLEEPDEEP	RW	Low power mode of the control system. 1: deepsleep 0: sleep	0
1	SLEEPONEXIT	RW	System status after control leaves the interrupt service program. 1: The system enters low-power mode. 0: The system enters the main program.	0
0	Reserved	RO	Reserved	0

3.2 Interrupt-related CSR Registers

In addition, the following CSR registers also have a significant impact on the processing of interrupts.

Interrupt System Control Register (INTSYSCR)

Name	CSR Address	Access	Description	Reset value
INTSYSCR	0x804	MRW	Interrupt system control register	0x00000000

Its folks are defined as.

Bit	Name	Access	Description	Reset value
[31:16]	Reserved	MRO	Reserved	0
[15:8]	PMTSTA	MRO	Preemption status indication. 0x00: no preemption bits in the priority configuration bits, no interrupt nesting occurs. 0x80: the highest bit in the priority configuration bit is a preemption bit, with 2 levels of interrupt nesting. 0xC0: priority configuration bits in which	0

			the high 2 bits are preempted and 4 levels of interrupts are nested. 0xE0: The high 3 bits of the priority configuration bits are preempted, with 8 levels of interrupt nesting. *Note: This status is valid only for V4F.*	
[7:6]	Reserved	MRO	Reserved	0
5	GIHWSTKNEN	MRW1	Global interrupt and *HPE* off enable. *Note: This bit is often used in real-time operating systems, when the interrupt switches context, set this bit to turn off the global interrupt and HPE out stack, when the context switch is complete, after the execution of the interrupt return, the hardware automatically clears this bit.*	0
4	HWSTKOVEN	MRW	Interrupt enable after *HPE* overflow. 0: Global interrupts are turned off after a HPE overflow. 1: Interrupts are still executable after a hardware stack overflow. *Note: HPE depth is 3. When the configuration nesting level is greater than 3, if the bit is set to 1, the low priority three interrupts need to be configured as HPE and the high priority as SPE.*	0
[3:2]	PMTCFG	MRW	Interrupt nesting depth configuration. 0b00: No nesting, the number of preemption bits is 0. 0b01: 2 levels of nesting, with 1 number of preemption bits. 0b10: 4 levels of nesting, with 2 preemption bits. 0b11: 8 levels of nesting, the number of preemption bits is 3. *Note: This status is valid only for V4F.*	0
1	INESTEN	MRW	Interrupt nesting enable. 0: Interrupt nesting function off. 1: Interrupt nesting function enabled.	0
0	HWSTKEN	MRW	HPE enable. 0: HPE function off. 1: HPE function enabled.	0

Machine Mode Exception Base Address Register (mtvec)

Name	CSR Address	Access	Description	Reset value
mtvec	0x305	MRW	Exception base address register	0x00000000

Its folks are defined as.

Bit	Name	Access	Description	Reset value
[31:2]	BASEADDR[31:2]	MRW	The interrupt vector table base address	0
1	MODE1	MRW	Interrupt vector table identifies patterns. 0: Identification by jump instruction, limited range, support for non-jump instructions. 1: Identify by absolute address, support full range, but must jump.	0
0	MODE0	MRW	Interrupt or exception entry address mode selection. 0: Use of the uniform entry address. 1: Address offset based on interrupt number *4.	0

For MCU of V4 series microprocessors, MODE[1:0]=11 is configured in the startup file by default, i.e. the vector table uses the absolute address of the interrupt function and the entry of the exception or interrupt is offset according to the interrupt number *4.

3.3 Interrupt Nesting

In conjunction with the Interrupt System Control Register INTSYSCR (CSR address: 0x804) and the Interrupt Priority Register PFIC_IPRIOR, nesting of interrupts can be allowed to occur. Enable nesting and configure the nesting depth in the interrupt system control register (V4 series MCUs are configured in the startup file), and configure the priority of the corresponding interrupt. The smaller the priority value, the higher the priority. The smaller the value of the preemption bit, the higher the preemption priority. If there are interrupts hanging at the same time under the same preemption priority, the microprocessor responds to the interrupt with the lower priority value (higher priority) first.

3.4 Hardware Prologue/Epilogue (HPE)

When an exception or interrupt occurs, the microprocessor stops the current program flow and shifts to the execution of the exception or interrupt handling function, the site of the current program flow needs to be saved. After the exception or interrupt returns, it is necessary to restore the site and continue the execution of the stopped program flow. For V4 series microprocessors, the "site" here refers to all the Caller Saved registers in Table 1-2.

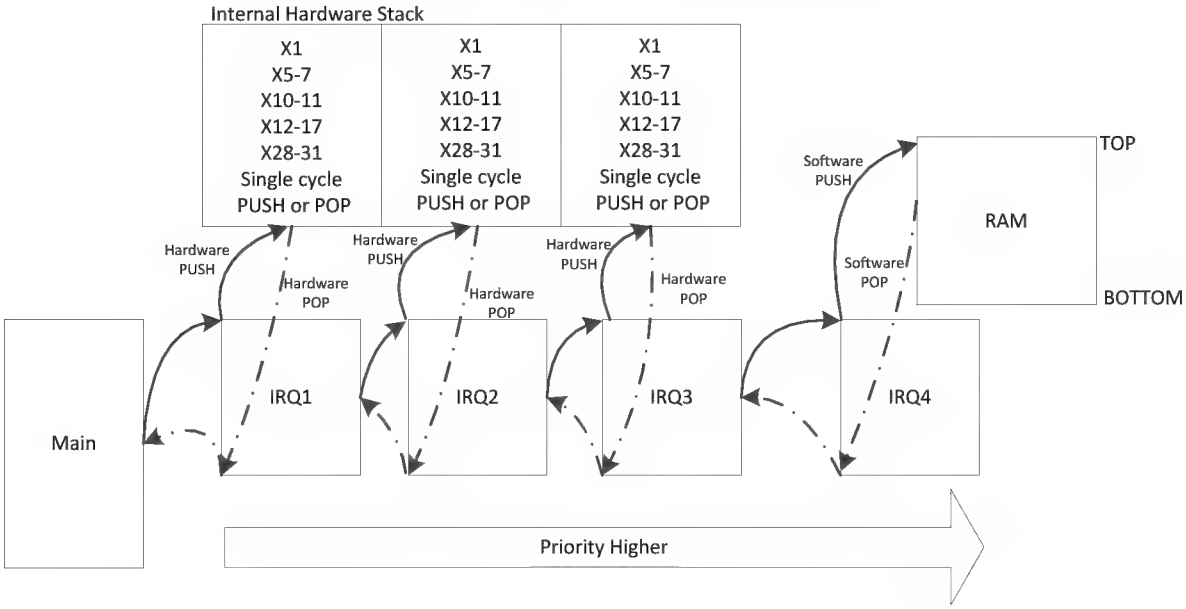
The V4 series microprocessors support hardware single cycle automatic saving of 16 of the shaped Caller Saved registers to an internal stack area that is not visible to the user. When an exception or interrupt returns, the hardware single cycle automatically restores data from the internal stack area to the 16 shaped registers. The hardware stack supports nesting with a maximum nesting depth of 3 levels. After a hardware stack overflow, if a higher priority interrupt is still allowed to execute, the "field" is saved to the user stack area.

When the nesting depth of interrupts allowed by the configuration is greater than the hardware stack depth, the interrupt response can be turned off after the hardware stack overflow is set by bit 4 of the interrupt system control register INTSYSCR, i.e., the maximum nesting depth is 3 levels, all using the hardware stack. If the interrupt continues to execute after allowing the hardware stack overflow, the priority of the interrupt

function using hardware stack needs to be set to the lowest three levels.

A schematic of the microprocessor pressure stack is shown in the following figure.

Figure 3-1 Schematic diagram of pressure stack



Note: 1. Hardware pressure stack depth may vary from model to model.

2.Interrupt functions using the hardware stack need to be compiled using MRS or its provided toolchain and the interrupt functions need to be declared with__attribute__((interrupt("WCH-Interrupt-fast"))).

3.If hardware floating point is used, the interrupt function with hardware stack declaration, the floating-point registers are still saved and restored by the compiler in software, and they are saved to the user stack area (RAM).

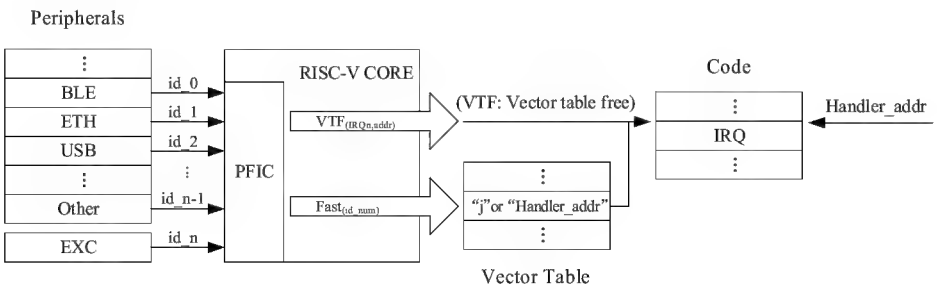
4.The interrupt function using stack push is declared by__attribute__((interrupt())).

3.5 Vector Table Free (VTF)

The Programmable Fast Interrupt Controller (PFIC) provides two VTF channels, i.e., direct access to the interrupt function entry without going through the interrupt vector table lookup process.

The PFIC responds to fast interrupts and VTF as shown in Figure 3-2 below.

Figure 3-2 Schematic diagram of programmable fast interrupt controller



Chapter 4 Physical Memory Protection (PMP)

In order to improve system security, the V4 series microprocessors are designed with Physical Memory Protection (PMP) modules in accordance with the RISC-V architecture standard. It supports access rights management for up to 4 physical regions. The PMP module is always in effect in user mode, and optionally in machine mode through the Lock (L) attribute.

The PMP module includes four sets of 8-bit configuration registers (one set of 32-bit) and four sets of address registers, all of which need to be accessed in machine mode using the CSR instruction.

Note: The number of protected areas supported by PMP may vary from one microprocessor model to another, as well as the number supported by the pmpcfg<n> and pmpaddr<i> registers, as detailed in Table 1-1.

4.1 PMP Register Sets

The list of CSR registers supported by the V4 microprocessor PMP module is shown in Table 4-1 below.

Table 4-1 PMP module register sets

Name	CSR address	Access	Description	Reset value
pmpcfg0	0x3A0	MRW	PMP configuration register 0	0x00000000
pmpaddr0	0x3B0	MRW	PMP address register 0	0x00000000
pmpaddr1	0x3B1	MRW	PMP address register 1	0x00000000
pmpaddr2	0x3B2	MRW	PMP address register 2	0x00000000
pmpaddr3	0x3B3	MRW	PMP address register 3	0x00000000

4.2 pmp<i>cfg Register

pmpcfg<n>, the configuration registers of the PMP unit, each register contains four 8-bit pmp<i>cfg fields corresponding to the configuration of the four regions. pmp<i>cfg indicates the configuration value of region i. The format is shown in Table 4-2 below.

Table 4-2 pmpcfg0 register

31 24	23 16	15 8	7 0	
pmp3cfg	pmp2cfg	pmp1cfg	pmp0cfg	pmpcfg0

The pmp<i>cfg is used to configure area i. Its bit definitions are described in detail in Table 4-3 below.

Table 4-3 pmp<i>cfg Register

Bit	Name	Description
7	L	Lock enable, unlockable in Machine mode 0: Not locked. 1: Lock the relevant register.
[6:5]	-	Reserved
[4:3]	A	Address alignment and protection area range selection. 00: OFF (PMP off) 01: TOR (Top Alignment Protection) 10: NA4 (fixed four-byte protection) 11: NAPOT ($2^{(G+2)}$ bytes protected, G≥1)
2	X	Executable property.

		0: No execute permission 1: Have execute permission
1	W	Writable property. 0: No write permission 1: Have write permission
0	R	Readable Properties 0: No read permission 1: Have read permission

4.3 pmpaddr<i> Register

The pmpaddr<i> register is used to configure the address of area i. Standardly defined in the RV32 architecture, it is the encoding of the high 32 bits of a 34-bit physical address in the format shown in Table 4-4 below. the entire physical address space of the V4 microprocessor is 4G, so the high two bits of this register are not used.

Table 4-4 pmpaddr<i> registers

31	0
address[33:2]	

When NAPOT is selected, the low bit of the address register is also used to indicate the size of the current protection area, as shown in the table below, where "y" is a bit of the register.

Table 4-5 PMP configuration and address registers and protection area relationship table

pmpaddr	pmpcfg.A	Matching base address and size
yyyy…yyyy	NA4	Base address "yy…yyyy00", 4-byte area protection
yyyy…yyy0	NAPOT	Base address "yy…yyyy000", 8- byte area protection
yyyy…yy01	NAPOT	Base address "yy…yyyy0000", 16- byte area protection
yyyy…y011	NAPOT	Base address "yy…yyyy00000", 32- byte area protection
…	…	…
yyy01…111	NAPOT	Base address "y0…000000", 2^{31}- byte area protection
yy011…111	NAPOT	The entire 232-byte area is protected

4.4 Protection Mechanism

X/W/R in pmp<i>cfg is used to set the protection permissions for region i. Violation of the relevant permissions will cause the corresponding exceptions.

(1) When an attempt is made to fetch an instruction in a PMP area that does not have execution privileges, a fetch instruction access error exception (mcause=1) will be raised.

(2) When attempting to write data in a PMP area without write permission, a store instruction access error exception (mcause=7) will be raised.

(3) When attempting to read data in a PMP area without read-out permission, a load instruction access error exception (mcause=5) will be raised.

A in pmp<i>cfg is used to set the protection range and address alignment for region i. For the A_ADDR ≤ region<i> < B_ADDR region for memory protection (both A_ADDR and B_ADDR are required to be 4-byte aligned).

(1) If B_ADDR - A_ADDR == 2^2, then the NA4 method is used.

(2) If B_ADDR - A_ADDR == $2^{(G+2)}$, G ≥ 1, and A_ADDR is $2^{(G+2)}$ aligned then the NAPOT method is

used.

(3) Otherwise, the TOP method is used.

Table 4-6 PMP address matching method

A value	Name	Description
0b00	OFF	No area to protect
0b01	TOR	Top Aligned Area Protection. pmp<i> cfg under pmpaddri-1 ≤ region<i> <pmpaddri; $pmpaddr_{i-1}$ = A_ADDR >> 2. $pmpaddr_i$ = B_ADDR >> 2. *Note: If area 0 of PMP is configured as TOR mode (i=0), the lower boundary of the protection area is 0 address, i.e. 0 ≤ addr < pmpaddr0, all within the matching range.*
0b10	NA4	Fixed 4-byte area protection. pmp<i>cfg under pmpaddri as the starting address of the 4-byte pmpaddri = A_ADDR>>2.
0b11	NAPOT	Protect the $2^{(G+2)}$ region with G ≥ 1, when A_ADDR is $2^{(G+2)}$ aligned. pmpaddri = ((A_ADDR|($2^{(G+2)}$-1)) &~(1<<G+1))>>2.

The L bit in pmp<i>cfg is used to lock the PMP entry. After locking, the configuration register pmp<i>cfg and the address register pmpaddr<i> will not be able to be modified. If A in pmp<i>cfg is set to TOR mode, pmpaddr<i-1> will also not be modified. when L is set, the X/W/R permissions defined in pmp<i>cfg are also valid for machine mode, and when L is cleared, X/W/R is only valid for user mode, and L is cleared only after system reset.

QingKe V4 series microprocessors support protection of multiple zones. When the same operation matches multiple zones at the same time, the zone with the smaller number is matched first.

Chapter 5 System Timer (SysTick)

QingKe V4 series microprocessor is designed with a 64-bit plus counter (SysTick) inside, and its clock source can be the system clock or 8 divisions of the system clock. It can provide time base for real time operating system, provide timing, measure time, etc. The timer involves 6 registers and maps to the peripheral address space for controlling the SysTick, as shown in Table 5-1 below.

Table 5-1 SysTick register list

Name	Access address	Description	Reset value
STK_CTLR	0xE000F000	System count control register	0x00000000
STK_SR	0xE000F004	System count status register	0x00000000
STK_CNTL	0xE000F008	System counter low register	0x00000000
STK_CNTH	0xE000F00C	System counter high register	0x00000000
STK_CMPLR	0xE000F010	System count comparison value low register	0x00000000
STK_CMPHR	0xE000F014	System count comparison value high register	0x00000000

Each register is described in detail as follows.

System Count Control Register (STK_CTLR)

Table 5-2 SysTick control registers

Bit	Name	Access	Description	Reset value
31	SWIE	RW	Software interrupt trigger enable (SWI). 1: Triggering software interrupts. 0: Turn off the trigger. After entering software interrupt, software clear 0 is required, otherwise it is continuously triggered.	0
[30:6]	Reserved	RO	Reserved	0
5	INIT	W1	Counter initial value update. 1: Updated to 0 on up counts, updated to the comparison value on down counts. 0: Invalid.	
4	MODE	RW	Counting mode. 1: Counting down. 0: Counting up.	
3	STRE	RW	Auto Reload Count enable bit. 1: Re-counting from 0 after counting up to the comparison value, and re-counting from the comparison value after counting down to 0. 0: Continue counting up/down.	
2	STCLK	RW	Counter clock source selection bit. 1: HCLK for time base. 0: HCLK/8 for time base.	
1	STIE	RW	Counter interrupt enable control bit.	

			1: Enable counter interrupts. 0: Turn off the counter interrupt.	
0	STE	RW	System counter enable control bit. 1: Start the system counter STK. 0: Turn off the system counter STK and the counter stops counting.	

System Count Status Register (STK_SR)

Table 5-3 SysTick status register

Bit	Name	Access	Description	Reset value
[31:1]	Reserved	RO	Reserved	0
0	CNTIF	RW	Counting value comparison flag, write 0 clear, write 1 invalid: 1: Count up to reach the comparison value and count down to 0. 0: The comparison value is not reached.	0

System Counter Low Register (STK_CNTL)

Table 5-4 SysTick counter low register

Bit	Name	Access	Description	Reset value
[31:0]	CNTL	RW	The current counter count value is 32 bits lower.	0

Note: The register STK_CNTL and the register STK_CNTH together constitute the 64-bit system counter.

System Counter High Register (STK_CNTH)

Table 5-5 SysTick counter high register

Bit	Name	Access	Description	Reset value
[31:0]	CNTH	RW	The current counter count value is 32 bits higher.	0

Note: The register STK_CNTL and the register STK_CNTH together constitute the 64-bit system counter.

System Count Comparison Value Low Register (STK_CMPLR)

Table 5-6 SysTick count comparison value low register

Bit	Name	Access	Description	Reset value
[31:0]	CMPL	RW	Set the counter comparison value 32 bits lower.	0

Note: The register STK_CMPLR and the register STK_CMPHR together constitute the 64-bit counter comparison value.

System Count Comparison Value High Register (STK_CMPHR)

Table 5-7 SysTick count comparison value high register

Bit	Name	Access	Description	Reset value
[31:0]	CMPH	RW	Set the counter comparison value 32 bits higher.	0

Note: The register STK_CMPLR and the register STK_CMPHR together constitute the 64-bit counter comparison value.

Chapter 6 Processor Low-power Settings

QingKe V4 series microprocessors support sleep state via WFI (Wait For Interrupt) instruction to achieve low static power consumption. Together with PFIC's system control register (PFIC_SCTLR), various Sleep modes and WFE instructions can be implemented.

6.1 Enter Sleep

QingKe V4 series microprocessors can go to sleep in two ways, Wait for Interrupt (WFI) and Wait For Event (WFE). The WFI method means that the microprocessor goes to sleep, waits for an interrupt to wake up, and then wakes up to the corresponding interrupt to execute. The WFE method means that the microprocessor goes to sleep, waits for an event to wake up, and wakes up to continue executing the previously stopped program flow.

The standard RISC-V supports WFI instruction, and the WFI command can be executed directly to enter sleep by WFI method. For the WFE method, the WFITOWFE bit in the system control register PFIC_SCTLR is used to control the subsequent WFI commands as WFE processing to achieve the WFE method to enter sleep.

The depth of sleep is controlled according to the SLEEPDEEP bit in PFIC_SCTLR.

- If the SLEEPDEEP in PFIC_SCTLR register is cleared to zero, the microprocessor enters Sleep mode and the internal unit clock is allowed to be turned off except for SysTick and part of the wake-up logic.
- If SLEEPDEEP in the PFIC_SCTLR register is set, the microprocessor enters Deep sleep mode and all cell clocks are allowed to be turned off.

When the microprocessor is in Debug mode, it is not possible to enter any kind of Sleep mode.

6.2 Sleep Wakeup

QingKe V4 series microprocessors can be woken up after sleep due to WFI and WFE in the following ways.

- After the WFI method goes to sleep, it can be awakened by

(1) The microprocessor can be woken up by the interrupt source responded by the interrupt controller. After waking up, the microprocessor executes the interrupt function first.

(2) Enter Sleep mode, debug request can make the microprocessor wake up and enter deep sleep, debug request cannot wake up the microprocessor.

- After going to sleep in the WFE mode, the microprocessor can be woken up by the following.

(1) Internal or external events, when there is no need to configure the interrupt controller, wake up and continue to execute the program.

(2) If an interrupt source is enabled, the microprocessor is woken up when an interrupt is generated, and after waking up, the microprocessor executes the interrupt function first.

(3) If the SEVONPEND bit in PFIC_SCTLR is configured, the interrupt controller does not enable the interrupt under, but when a new interrupt pending signal is generated (the previously generated pending signal does not take effect), it can also make the microprocessor wake up, and the corresponding interrupt pending flag needs to be cleared manually after waking up.

(4) Enter Sleep mode debug request can make the microprocessor wake up and enter deep sleep, debug

request cannot wake up the microprocessor.

In addition, the state of the microprocessor after wake-up can be controlled by configuring the SLEEPONEXIT bit in PFIC_SCTLR.

- SLEEPONEXIT is set and the last level interrupt return instruction (mret) will trigger the WFI mode sleep.

- SLEEPONEXIT is cleared with no effect.

Various MCU products equipped with V4 series microprocessors can adopt different sleep modes, turn off different peripherals and clocks, implement different power management policies and wake-up methods according to different configurations of PFIC_SCTLR, and realize various low-power modes.

Chapter 7 Debug Support

QingKe V4 series microprocessors include a hardware debug module that supports complex debugging operations. When the microprocessor is suspended, the debug module can access the microprocessor's GPRs, CSRs, Memory, external devices, etc. through abstract commands, program buffer deployment instructions, etc. The debug module can suspend and resume the microprocessor's operation.

The debug module follows the RISC-V External Debug Support Version0.13.2 specification, detailed documentation can be downloaded from RISC-V International website.

7.1 Debug Module

The debug module inside the microprocessor, capable of performing debug operations issued by the debug host, includes.

- Access to registers through the debug interface
- Reset, suspend and resume the microprocessor through the debug interface
- Read and write memory, instruction registers and external devices through the debug interface
- Deploy multiple arbitrary instructions through the debug interface
- Set software breakpoints through the debug interface
- Set hardware breakpoints through the debug interface
- Support for automatic execution of abstract commands
- Support single-step debugging

Note: Hardware breakpoints are only supported by the V4C, V4F, and V4J microprocessors.

The internal registers of the debugging module use a 7-bit address code, and the following registers are implemented inside QingKe V4 series microprocessors.

Table 7-1 Debug module register List

Name	Access address	Description
data0	0x04	Data register 0, can be used for temporary storage of data
data1	0x05	Data register 1, can be used for temporary storage of data
dmcontrol	0x10	Debug module control register
dmstatus	0x11	Debug module status register
hartinfo	0x12	Microprocessor status register
abstractcs	0x16	Abstract command status register
command	0x17	Abstract command register
progbuf0-7	0x20-0x27	Instruction cache registers 0-7
haltsum0	0x40	Pause status register

The debug host can control the microprocessor's suspend, resume, reset, etc. by configuring the dmcontrol register. The RISC-V standard defines three types of abstract commands: access register, fast access, and access memory. QingKe V4 microprocessor supports two of them, and does not support fast access. The abstract commands can be used to access registers (GPRs, CSRs, FPRs), sequential access to memory, etc.

The debug module implements eight instruction cache registers progbuf0-7, and the debug host can cache multiple instructions (which can be compressed instructions) to the buffer, and can choose to continue executing the instructions in the instruction cache registers after executing the abstract command, or execute the cached instructions directly. Note that the last instruction in the progbufs needs to be an "ebreak" or "c.ebreak" instruction. Access to storage, peripherals, etc. is also possible through abstract commands and instructions cached in the progbufs.

Each register is described in detail as follows.

Data Register 0 (data0)

Table 7-2 data0 register definition

Bit	Name	Access	Description	Reset Value
[31:0]	data0	RW	Data register 0, used for temporary storage of data	0

Data Register 1 (data1)

Table 7-3 data1 register definition

Bit	Name	Access	Description	Reset Value
[31:0]	data1	RW	Data register 1, used for temporary storage of data	0

Debug Module Control Register (dmcontrol)

This register controls the pause, reset, and resume of the microprocessor. Debug host write data to the corresponding field to achieve pause (haltreq), reset (ndmreset), resume (resumereq). You describe into the following.

Table 7-4 dmcontrol register definition

Bit	Name	Access	Description	Reset Value
31	haltreq	WO	0: Clear the pause request 1: Send a pause request	0
30	resumereq	W1	0: Invalid 1: Restore the current microprocessor *Note: Write 1 is valid and the hardware is cleared after the microprocessor is recovered*	0
29	Reserved	RO	Reserved	0
28	ackhavereset	W1	0: Invalid 1: Clear the haverest status bit of the microprocessor	0
[27:2]	Reserved	RO	Reserved	0
1	ndmreset	RW	0: Clear reset 1: Reset the entire system other than the debug module	0
0	dmactive	RW	0: Reset debug module 1: Debug module works properly	0

Debug Module Status Register (dmstatus)

This register is used to indicate the status of the debug module and is a read-only register with the following

description of each bit.

Table 7-5 dmstatus register definition

Bit	Name	Access	Description	Reset Value
[31:20]	Reserved	RO	Reserved	0
19	allhavereset	RO	0: Invalid 1: Microprocessor reset	0
18	anyhavereset	RO	0: Invalid 1: Microprocessor reset	0
17	allresumeack	RO	0: Invalid 1: Microprocessor reset	0
16	anyresumeack	RO	0: Invalid 1: Microprocessor reset	0
[15:14]	Reserved	RO	Reserved	0
13	allavail	RO	0: Invalid 1: Microprocessor is not available	0
12	anyavail	RO	0: Invalid 1: Microprocessor is not available	0
11	allrunning	RO	0: Invalid 1: Microprocessor is running	0
10	anyrunning	RO	0: Invalid 1: Microprocessor is running	0
9	allhalted	RO	0: Invalid 1: Microprocessor is in suspension	0
8	anyhalted	RO	0: Invalid 1: Microprocessor out of suspension	0
7	authenticated	RO	0: Authentication is required before using the debug module 1: The debugging module has been certified	0x1
[6:4]	Reserved	RO	Reserved	0
[3:0]	version	RO	Debugging system support architecture version 0010: V0.13	0x2

Microprocessor Status Register (hartinfo)

This register is used to provide information about the microprocessor to the debug host and is a read-only register with each bit described as follows.

Table 7-6 hartinfo register definition

Bit	Name	Access	Description	Reset Value
[31:24]	Reserved	RO	Reserved	0
[23:20]	nscratch	RO	Number of dscratch registers supported	0x3
[19:17]	Reserved	RO	Reserved	0
16	dataaccess	RO	0: Data register is mapped to CSR address 1: Data register is mapped to memory address	0x1
[15:12]	datasize	RO	Number of data registers	0x2
[11:0]	dataaddr	RO	Data register data0 offset address, the base address is 0xe0000000	0x380

Abstract Command Control and Status Registers (abstractcs)

This register is used to indicate the execution of the abstract command. The debug host can read this register to know whether the last abstract command is executed or not, and can check whether an error is generated during the execution of the abstract command and the type of the error, which is described in detail as follows.

Table 7-7 abstractcs register definitions

Bit	Name	Access	Description	Reset Value
[31:29]	Reserved	RO	Reserved	0
[28:24]	progbufsize	RO	Indicates the number of program buffer program cache registers	0x8
[23:13]	Reserved	RO	Reserved	0
12	busy	RO	0: No abstract command is executing 1: There are abstract commands being executed *Note: After execution, the hardware is cleared.*	0
11	Reserved	RO	Reserved	0
[10:8]	cmder	RW	Abstract command error type 000: No error 001: abstract command execution to write to command, abstractcs, abstractauto registers or read and write to data and progbuf registers 010: Does not support current abstract command 011: Execution of abstract command with exception 100: The microprocessor is not suspended or unavailable and cannot execute abstract commands 101: Bus error 110: Parity bit error during communication 111: Other errors *Note: For bit writing 1 is used to clear the zero.*	0
[7:4]	Reserved	RO	Reserved	0
[3:0]	datacount	RO	Number of data registers	0x2

Abstract Command Register(command)

The debug host can access the GPRs, FPRs, and CSRs registers inside the microprocessor by writing different configuration values in the abstract command registers.

When accessing the registers, the command register bits are defined as follows.

Table 7-8 Definition of command register when accessing registers

Bit	Name	Access	Description	Reset Value
[31:24]	cmdtype	WO	Abstract command type 0: Access register 1: Quick access (not supported) 2: Access to memory (not supported)	0

23	Reserved	WO	Reserved	0
[22:20]	aarsize	WO	Access register data bit width 000: 8-bit 001: 16-bit 010: 32-bit 011: 64-bit (not supported) 100: 128-bit (not supported) *Note: When accessing floating-point registers FPRs, only 32-bit access is supported.*	0
19	aarpostincrement	WO	0: No effect 1: Automatically increase the value of regno after accessing the register	0
18	postexec	WO	0: No effect 1：Execute the abstract command and then execute the command in progbuf	0
17	transfer	WO	0: Do not execute the operation specified by write 1: Execute the manipulation specified by write	0
16	write	WO	0: Copy data from the specified register to data0 1: Copy data from data0 register to the specified register	0
[15:0]	regno	WO	Specify access registers 0x0000-0x0fff are CSRs 0x1000-0x101f are GPRs 0x1020-0x103f are FPRs	0

When accessing the memory, the command register bits are defined as follows.

Table 7-9 Definition of command register when accessing memory

Bit	Name	Access	Description	Reset Value
[31:24]	cmdtype	WO	Abstract command type 0: Access register 1: Quick access (not supported) 2: Access to memory	0
23	aamvirtual	WO	0: Access to physical address 1: Access to virtual addresses	0
[22:20]	aarsize	WO	Access register data bit width 000: 8-bit 001: 16-bit 010: 32-bit 011: 64-bit (not supported) 100: 128-bit (not supported)	0
19	aarpostincrement	WO	0: No effect 1: The address of the data1 register is incremented by the number of bytes corresponding to the bit width of the aamsize configuration after successful access to memory	0

			aamsize=0, access by byte, data1 plus 1 aamsize=1, by half-word range, data1 plus 2 aamsize=2, access by word, data1 plus 4	
18	postexec	WO	0: No effect 1：Execute the abstract command and then execute the command in progbuf	0
17	Reserved	RO	Reserved	0
16	write	WO	0: Copy data from the specified register to data0 1: Copy data from data0 register to the specified register	0
[15:14]	target-specific	WO	Read and write method definition For writing. 00, 01: Write directly to memory 10: write the result to memory after the data in data0 and the data bits in the memory or (only word access is supported) 11: write the result to memory after the data in data0 and the data bits in memory (only word access is supported) For reading. 00, 01, 10, 11: Read directly to memory	0
[13:0]	Reserved	RO	Reserved	0

Abstract Command Auto-execution Register (abstractauto)

This register is used to configure the debug module so that abstract commands can be executed again when reading and writing to the progbufx and datax of the debug module, which is described as follows.

Table 7-10 abstractauto register definition

Bit	Name	Access	Description	Reset Value
[31:16]	autoexecprogbuf	RW	If a location 1, the corresponding read or write to progbufx will cause the abstract command in the command register to be executed again *Note: The V4 series is designed with 8 progbuf, corresponding to bits [23:16]*	0
[15:12]	Reserved	RO	Reserved	0
[11:0]	autoexecdata	RW	If a position 1, the corresponding read or write to the datax register will cause the abstract command in the command register to be executed again *Note: V4 series design 2 data registers, corresponding to bits [1:0]*	0

Instruction Cache Register (progbufx)

This register is used to store any instruction, deploy the corresponding operation, including 8, need to pay attention to the last execution needs to be "ebreak" or "c.ebreak".

Table 7-11 progbuf register definition

Bit	Name	Access	Description	Reset Value
[31:0]	progbuf	RW	Instruction encoding for cache operations, which may include compression instructions	0

Pause Status Register (haltsum0)

This register is used to indicate whether the microprocessor is suspended or not. Each bit indicates the suspended status of a microprocessor, and when there is only one core, only the lowest bit of this register is used to indicate it.

Table 7-12 haltsum0 register definition

Bit	Name	Access	Description	Reset Value
[31:1]	Reserved	RO	Reserved	0
0	haltsum0	RO	0: Microprocessor operates normally 1: Microprocessor stop	0

In addition to the above-mentioned registers of the debug module, the debug function also involves some CSR registers, mainly the debug control and status register dcsr and the debug instruction pointer dpc, which are described in detail as follows.

Debug Control and Status Register (dcsr)

Table 7-13 dcsr register definition

Bit	Name	Access	Description	Reset Value
[31:28]	xdebugver	DRO	0000: External debugging is not supported 0100: Support standard external debugging 1111: External debugging is supported, but does not meet the specification	0x4
[27:16]	Reserved	DRO	Reserved	
15	ebreakm	DRW	0: The ebreak command in Machine mode behaves as described in the privilege file 1: The ebreak command in Machine mode can enter Debug mode	0
[14:13]	Reserved	DRO	Reserved	0
12	ebreaku	DRW	0: The ebreak command in User mode behaves as described in the privilege file 1: The ebreak command in User mode can enter debug mode	0
11	stepie	DRW	0: Interrupts are disabled under single-step debugging 1: Enable interrupts under single-step debugging	0
10	Reserved	DRO	Reserved	0
9	stoptime	DRW	0: System timer running in Debug mode 1: System timer stop in Debug mode	0
[8:6]	cause	DRO	Reasons for entering debugging 001: Entering debugging in the form of ebreak command (priority 3)	0

			010: Entering debugging in the form of trigger module (priority 4, the highest) 011: Entering debugging in the form of pause request (priority 1) 100: debugging in the form of single-step debugging (priority 0, the lowest) 101: enter debug mode directly after microprocessor reset (priority 2) Others: Reserved	
[5:3]	Reserved	DRO	Reserved	0
2	step	DRW	0: Turn off single-step debugging 1: Enable single-step debugging	0
[1:0]	prv	DRW	Privilege mode 00: User mode 01: Supervisor mode (not supported) 10: Reserved 11: Machine mode *Note: Record the privileged mode when entering debug mode, the debugger can modify this value to modify the privileged mode when exiting debug*	0

Debug Mode Program Pointer (dpc)

This register is used to store the address of the next instruction to be executed after the microprocessor enters Debug mode, and its value is updated with different rules depending on the reason for entering debug. dpc register is described in detail as follows.

Table 7-14 dpc register definitions

Bit	Name	Access	Description	Reset Value
[31:0]	dpc	DRW	Instruction address	0

The rules for updating the registers are shown in the following table.

Table 7-15 dpc update rules

Enter the debugging method	dpc Update rules
ebreak	Address of the Ebreak instruction
single step	Instruction address of the next instruction of the current instruction
trigger module	Temporarily not supported
halt request	Address of the next instruction to be executed when entering Debug

7.2 Debug Interface

Different from the standard RISC-V defined JTAG interface, QingKe V4 series microprocessor using single / two-wire debugging interface, follow the WCH debugging interface protocol. The two-wire interface of V4A follows the interface protocol V1.0, while V4B, V4C, V4F, V4J follow the interface protocol V1.1. The debugging interface is responsible for the communication between the debug host and the debug module, and realizes the read/write operation of the debug host on the registers of the debug module. WCH designed WCH_Link and open source its schematic and program binary files, which can be used for debugging all

RISC-V architecture microprocessors.

The specific debugging interface protocol refers to WCH debugging protocol manual.

Note: Only V4J supports both 1-wire and 2-wire debug interfaces.

Chapter 8 CSR Register Lists

The RISC-V architecture defines a number of Control and Status Registers (CSRs) for controlling and recording the operating status of the microprocessor. Some of the CSRs have been introduced in the previous section, and this chapter will detail the CSR registers implemented in the QingKe V4 series microprocessors.

8.1 CSR Register Lists

Table 8-1 List of Microprocessor CSR Registers

Type	Name	CSR Address	Access	Description
RISC-V Standard CSR	marchid	0xF12	MRO	Architecture number register
	mimpid	0xF13	MRO	Hardware implementation numbering register
	mstatus	0x300	MRW	Status register
	misa	0x301	MRW	Hardware instruction set register
	mtvec	0x305	MRW	Exception base address register
	mscratch	0x340	MRW	Machine mode staging register
	mepc	0x341	MRW	Exception program pointer register
	mcause	0x342	MRW	Exception cause register
	mtval	0x343	MRW	Exception value register
	pmpcfg<i>	0x3A0+i	MRW	PMP unit configuration register
	pmpaddr<i>	0x3B0+i	MRW	PMP unit address register
	fflags	0x001	URW	Floating-point exception flag register
	frm	0x002	URW	Floating-point rounding mode register
	fcsr	0x003	URW	Floating-point control and status register
	dcsr	0x7B0	DRW	Debug control and status registers
	dpc	0x7B1	DRW	Debug mode program pointer register
	dscratch0	0x7B2	DRW	Debug mode staging register 0
	dscratch1	0x7B3	DRW	Debug mode staging register 1
Vendor-defined CSRs	gintenr	0x800	URW	Global interrupt enable register
	intsyscr	0x804	URW	Interrupt system control register
	corecfgr	0xBC0	MRW	Microprocessor configuration register
	cstrcr	0xBC2	MRW	Cache policy configuration register
	cpmpocr	0xBC3	MRW	Cache policy overrides PMP control registers
	cmcr	0xBD0	MWO	Cache operation control register
	cinfor	0xFC0	MRO	Cache information register

8.2 RISC-V Standard CSR Registers

Architecture Number Register (marchid)

This register is a read-only register to indicate the current microprocessor hardware architecture number, which is mainly composed of vendor code, architecture code, series code, and version code. Each of them is defined as follows.

Table 8-2 marchid register definition

Bit	Name	Access	Description	Reset Value
31	Reserved	MRO	Reserved	1
[30:26]	Vender0	MRO	Manufacturer code 0 Fixed to the letter "W" code	0x17

[25:21]	Vender1	MRO	Manufacturer code1 Fixed to the letter "C" code	0x03
[20:16]	Vender2	MRO	Manufacturer code 2 Fixed to the letter "H" code	0x08
15	Reserved	MRO	Reserved	1
[14:10]	Arch	MRO	Architecture code RISC-V architecture is fixed to the letter "V" code	0x16
[9:5]	Serial	MRO	Series code QingKe V4 series, fixed to the number "4"	0x04
[4:0]	Verision	MRO	Version code Can be the version "A", "B", "C", "F" and other letters of the code	x

The manufacturer number and version number are alphabetic, and the series number is numeric. The coding table of letters is shown in the following table.

Table 8-3 Alphabetic Mapping Table

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

For example, the QingKe V4F microprocessor, read the value of this register as: 0xDC68D886, which corresponds to WCH-V4F.

Hardware Implementation Numbering Register (mimpid)

This register is mainly composed of vendor codes, each of which is defined as follows.

Table 8-4 mimpid Register Definition

Bit	Name	Access	Description	Reset Value
31	Reserved	MRO	Reserved	1
[30:26]	Vender0	MRO	Manufacturer code 0 Fixed to the letter "W" code	0x17
[25:21]	Vender1	MRO	Manufacturer code1 Fixed to the letter "C" code	0x03
[20:16]	Vender2	MRO	Manufacturer code 2 Fixed to the letter "H" code	0x08
15	Reserved	MRO	Reserved	1
[14:1]	Reserved	MRO	Reserved	0
0	Reserved	MRO	Reserved	1

Machine Mode Status Register (mstatus)

This register has been partially described in the previous section, and its folks are positioned as follows.

Table 8-5 mstatus Register Definition

Bit	Name	Access	Description	Reset Value
[31:15]	Reserved	MRO	Reserved	0
[14:13]	Reserved	MRO	Floating-point unit status FS: FS Meaning 00: OFF 01: Initial	0

			10 Clean 11 Dirty	
[12:11]	MPP	MRW	Privileged mode before entering break	0
[10:8]	Reserved	MRO	Reserved	0
7	MPIE	MRW	Interrupt enable state before entering interrupt	0
[6:4]	Reserved	MRO	Reserved	0
3	MIE	MRW	Machine mode interrupt enable	0
[2:0]	Reserved	MRO	Reserved	0

The FS field is used to describe and maintain the floating-point unit state, so this field is only meaningful on the QingKe V4F microprocessor that contains the hardware floating point function. If the value is 0, it means that the floating-point unit is off, and if the floating-point instruction is used at this time, an exception will be triggered; if the value is 1 or 2, the field will be updated to 3 when the floating-point instruction is executed. if the user does not expect to use the hardware floating point function when using the V4F microprocessor, the two bits can be cleared manually in machine mode to turn off the hardware floating point and reduce power consumption.

The MPP field is used to save the privileged mode before entering the exception or interrupt, and is used to restore the privileged mode after exiting the exception or interrupt. MIE is the global interrupt enable bit, and when entering the exception or interrupt, the value of MPIE is updated to the value of MIE, and it should be noted that in the QingKe V4 series microprocessors, MIE will not be updated to 0 before the last level of nested interrupts to ensure that the interrupt nesting in machine mode continues to be executed. When an exception or interrupt is exited, the microprocessor reverts to the machine mode saved by MPP and the MIE is restored to the MPIE value.

QingKe V4 microprocessor supports Machine mode and User mode. If you need to make the microprocessor work only in Machine mode, you can set the MPP to 0x3 in the initialization of the startup file, i.e. after returning, it will always remain in Machine mode.

Hardware Instruction Set Register (misa)

This register is used to indicate the architecture of the microprocessor and the supported instruction set extensions, each of which is described as follows.

Table 8-6 misa Register Definition

Bit	Name	Access	Description	Reset Value
[31:30]	MXL	MRO	Machine word length 1:32 2:64 3:128	1
[29:26]	Reserved	MRO	Reserved	0
[25:0]	Extensions	MRO	Instruction set extensions	x

The MXL is used to indicate the word length of the microprocessor, QingKe V4 are 32-bit microprocessors, the domain is fixed to 1. Extensions are used to indicate that the microprocessor supports extended instruction set details, each indicates a class of extensions, its detailed description is shown in the following table.

Table 8-7 Instruction Set Extension Details

Bit	Name	Description
0	A	Atomic extension
1	B	Tentatively reserved for Bit-Manipulation extension
2	C	Compressed extension
3	D	Double-precision floating-point extension
4	E	RV32E base ISA
5	F	Single-precision floating-point extension
6	G	Additional standard extensions present
7	H	Hypervisor extension
8	I	RV32I/64I/128I base ISA
9	J	Tentatively reserved for Dynamically Translated Languages extension
10	K	Reserved
11	L	Tentatively reserved for Decimal Floating-Point extension
12	M	Integer Multiply/Divide extension
13	N	User-level interrupts supported
14	O	Reserved
15	P	Tentatively reserved for Packed-SIMD extension
16	Q	Quad-precision floating-point extension
17	R	Reserved
18	S	Supervisor mode implemented
19	T	Tentatively reserved for Transactional Memory extension
20	U	User mode implemented
21	V	Tentatively reserved for Vector extension
22	W	Reserved
23	X	Non-standard extensions present
24	Y	Reserved
25	Z	Reserved

For example, for the QingKe V4F microprocessor, the register value is 0x40901125, which means that the supported instruction set architecture is RV32IMACF, as well as the non-standard extension X, and has a user mode implementation.

Machine Mode Exception Base Address Register (mtvec)

This register is used to store the base address of the exception or interrupt handler and the lower two bits are used to configure the mode and identification method of the vector table as described in Section 3.2.

Machine Mode Staging Register (mscratch)

Table 8-8 mscratch register definitions

Bit	Name	Access	Description	Reset Value
[31:0]	mscratch	MRW	Data storage	0

This register is a 32-bit readable and writable register in machine mode for temporary data storage. For example, when entering an exception or interrupt handler, the user stack pointer SP is stored in this register and the interrupt stack pointer is assigned to the SP register. After exiting the exception or interrupt, restore the value of user stack pointer SP from mscratch. That is, the interrupt stack and user stack can be isolated.

Machine Mode Exception Program Pointer Register (mepc)

Table 8-9 mepc register definitions

Bit	Name	Access	Description	Reset Value
[31:0]	mepc	MRW	Exception procedure pointer	0

This register is used to save the program pointer when entering an exception or interrupt. It is used to save the instruction PC pointer before entering an exception when an exception or interrupt is generated, and mepc is used as the return address when the exception or interrupt is handled and used for exception or interrupt return. However, it is important to note that.

- When an exception occurs, mepc is updated to the PC value of the instruction currently generating the exception.
- When an interrupt occurs, mepc is updated to the PC value of the next instruction.

When you need to return an exception after processing the exception, you should pay attention to modifying the value of the mepc, and more details can be found in Chapter 2 Exceptions.

Machine Mode Exception Cause Register (mcause)

Table 8-10 mcause register definition

Bit	Name	Access	Description	Reset Value
31	Interrupt	MRW	Interrupt indication field 0: Exception 1: Interruption	0
[30:0]	Exception Code	MRW	Exception codes, see Table 2-1 for details	0

This register is mainly used to store the cause of the exception or the interrupt number of the interrupt. Its highest bit is the Interrupt field, which is used to indicate whether the current occurrence is an exception or an interrupt. The lower bit is the exception code, which is used to indicate the specific cause. Its details can be found in Chapter 2 Exceptions.

Machine Mode Exception Value Register (mtval)

Table 8-11 mtval register definition

Bit	Name	Access	Description	Reset Value
[31:0]	mtval	MRW	Exception value	0

This register is used to hold the value that caused the exception when an exception occurs. For details such as the value and time of its storage, please refer to Chapter 2 Exceptions.

PMP Configuration Register (pmpcfg<i>)

This register is mainly used for the configuration of the physical memory protection unit. Every 8 bits of this register is used to configure the protection of one area, refer to Chapter 4 for detailed definition.

PMP Configuration Register (pmpaddr<i>)

This register is mainly used for the address configuration of the physical memory protection unit, which is the encoding of the high 32 bits of a 34-bit physical address, refer to Chapter 4 for the specific configuration method.

Floating-point Control and Status Registers (fcsr)

This register exists only in microprocessors that support hardware floating point and is used to configure the rounding mode for floating point calculations and to record floating point exception flags. Each of its digits is defined as shown below.

Table 8-12 fcsr register definition

Bit	Name	Access	Description	Reset Value
[31:8]	Reserved	MRO	Reserved	0
[7:5]	FRM	MRW	Floating-point rounding mode	0
4	NV	MRW	Illegal operation exception	0
3	DZ	MRW	De-zeroing exception	0
2	OF	MRW	Upper overflow exception	0
1	UF	MRW	Under overflow exception	0
0	NX	MRW	Non-precision exception	0

It should be noted that an exception generated by the floating-point unit does not trigger an exception interrupt, but only sets the corresponding flag bit. The FRM field is used to configure the rounding mode of the floating-point unit, and the supported rounding modes are shown in the following table.

Table 8-13 Rounding Mode

Rounding code	Rounding mode	Description
000	RNE	Rounding to the nearest value, even values are preferred
001	RTZ	Rounding to zero
010	RDN	Rounding down (to -∞)
011	RUP	Rounding up (to ∞)
100	RMM	Round to the nearest value, first maximum
101	-	Illegal value
110	-	Illegal value
111	-	Dynamic rounding

Floating-point Status Registers (fflags)

This register exists only in microprocessors that support hardware floating point. This register is the exception flag bit field in fcsr, which is added in order to facilitate users to read and write exception flags directly and independently using CSR instructions.

Rounding Mode Register (frm)

This register exists only in microprocessors that support hardware floating-point. This register is the rounding mode field FRM in fcsr, which is added in order to facilitate the user to directly and independently configure the rounding mode for floating point calculations using CSR instructions.

Debug Control and Status Register (dcsr)

This register is used to control and record the operation status of Debug mode, refer to section 7.1 for detailed description.

Debug Mode Program Pointer Register (dpc)

This register is used to store the address of the next instruction to be executed after the microprocessor enters Debug mode, and its value is updated with different rules depending on the reason for entering debug. Refer to Section 7.1 for detailed description.

Debug Mode Staging Register (dscratch0-1)

This group of registers is used for temporary storage of data in Debug mode.

Table 8-14 dscratch0-1 register definitions

Bit	Name	Access	Description	Reset Value
[31:0]	dscratch	DRW	Debug mode data staging value	0

8.3 User-defined CSR Registers

Global Interrupt Enable Register (gintenr)

This register is used to control the enable and mask of global interrupt. The enable and mask of global interrupt in Machine mode can be controlled by the MIE and MPIE bits in register mstatus, which cannot be operated in User mode. The global interrupt enable register gintenr is the mapping of MIE and MPIE in mstatus, and can be used to set and clear MIE and MPIE by operating gintenr in User mode.

Note: Global interrupts do not include non-maskable interrupts NMI and exceptions

Interrupt System Control Register (intsyscr)

This register is mainly used to configure the interrupt nesting depth, HPE and other related functions, as described in Section 3.2.

Microprocessor Configuration Registers (corecfgr)

This register is mainly used to configure the microprocessor pipeline, instruction prediction and other related features, and generally does not need to be operated. The relevant MCU products are configured with default values in the startup file.

Cache Policy Configuration Register (cstrcr)

This register is mainly used to configure the policy and enable of the cache as shown in the following table:

Table 8-15 cstrcr register definitions

Bit	Name	Access	Description	Reset value
[31:26]	Reserve	MR0	Reserved	0
25	Icsramstren	MRW	Enable caching of instructions in the SRAM area 0: Disable caching of instructions in the SRAM area 1: Allow caching of instructions in the SRAM area	0
24	Iccodestren	MRW	Enable caching of instructions in the CODE area 0: Disable caching of instructions in the CODE area 1: Allow caching of instructions in the CODE area.	1
[23:2]	Reserve	MR0	Reserved	0
1	Icdisable	MRW	Command cache disabled 0: Enable command cache function 1: Disable command cache function	1
0	Reserve	MR0	Reserved	0

Cache Policy Override PMP Control Registers (cpmpocr)

When the instruction or data address and the address of the PMP channel control match, you can use this register to configure is to execute the corresponding cache policy or the control policy of the PMP channel that matches it. Every 4-bit group, a total of 8 groups, corresponding to a maximum of 8 PMP channels, the specific number of channels related to the corresponding specific implementation. The specific definitions are shown in the following table, where n denotes the PMP channel:

Table 8-16 cpmpocr register definitions

Bit	Name	Access	Description	Reset value
[4n+3:4n+1]	Reserve	MRO	Reserved	0
4n	IcPMPncach	MRW	Cache policy override PMP channel n policy enable 0: Disable cache policy override PMPn policy, execute the policy of the matching PMPn 1: Enable cache policy override PMPn policy, enforcing the policy configured in the cache policy configuration register cstrcr	0

Cache Operation Control Register (cmcr)

The cache operation control register is mainly used to clear or invalidate instructions or data in the cache.

Table 8-17 cmcr register definitions

Bit	Name	Access	Description	Reset value
[31:5]	Vaddr	MWO	Address or index information for the operation	0
[4:3]	Reserve	MRO	Reserved	0
2	IdxMode	MWO	Indexing mode 0: Execute the operation with the index information in the Vaddr value as the first address 1: Execute the operation with the Vaddr value as the address information	0
[1:0]	Opcode	MWO	Operation code 00: Disable instruction cache Other: Reserved	0

Cache Information Register (cinfor)

Used to indicate cache information, which is described in detail below:

Table 8-18 cinfor register definitions

Bit	Name	Access	Description	Reset value
[31:7]	Reserve	MRO	Reserved	0
[6:5]	Icway	MRO	Instruction cache ways 00: 1-way 01: 2-way 10: 4-way 11: 8-way	0
[4:2]	Icsize	MRO	Command cache capacity information 000: No cache 001: 4KB 010: 8KB 011: 16KB 100: 32KB	0

			101: 64KB Others: Reserved	
[1:0]	Iclinesize	MRO	Instruction cache line length 00: 8 bytes 01: 16 bytes 10: 32 bytes 11: 64 bytes	0